

2. Базовые понятия ООП

Суть объектно-ориентированного подхода

«Фундаментальная особенность нашего понимания мира заключается в том, что мы систематизируем свой жизненный опыт, представляя его в виде отдельных понятий или объектов (столы и стулья, банковские займы и алгебраические выражения, многочлены и люди, транзисторы и треугольники и т. п.), и наше мышление, язык и действия основываются на обозначении, описании и манипуляциях с этими объектами, с каждым в отдельности или в связи с другими объектами...

Всякий **объект**, представленный в запоминающем устройстве вычислительной машины в виде **записи** (record), будет обладать одним или более **атрибутами** (attributes), с которыми и приходится иметь дело при решении задачи...

Объекты реального мира часто удобно классифицировать с помощью определенного числа **классов** (classes), причем любой класс обозначается некоторым собирательным именем, таким как «человек», «банковский заем», «выражение» и т. д.»

Энтони Хоар, 1966 год.

Объект имеет имя и определяется:

- **Атрибутами/свойствами** (*размер, цвет, вес, ...*)
- **Поведениями/действиями** (*бежать, говорить, ...*)

House
- address - numberFloors - numberWindows
+ build() + destroy() + repair()

Human
- age - height - weight
+ speak() + walk() + sleep()

TV
- manufacturer - model - size
+ on() + switchChannels() + off()

Объект имеет имя и определяется:

- **Атрибутами/свойствами** (*размер, цвет, вес, ...*)
- **Поведениями/действиями** (*бежать, говорить, ...*)

House
- address - numberFloors - numberWindows
+ build() + destroy() + repair()

Human
- age - height - weight
+ speak() + walk() + sleep()

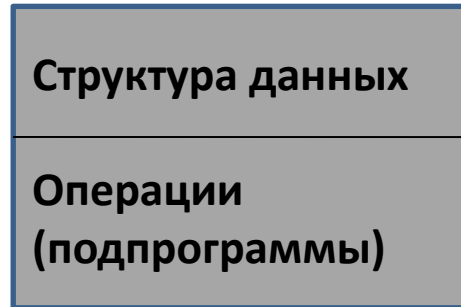
TV
- manufacturer - model - size
+ on() + switchChannels() + off()

Абстракция данных (реальных сущностей)

Объект в ООП — обладающий именем и физически находящийся в памяти компьютера набор **данных** (поля или свойства объекта) и **методов**, имеющих доступ к ним.

Принцип абстракции данных

Из сущности выделяются только необходимые характеристики и поведение. Данные и работающие с ними методы помещаются в один объект.



Объект = Данные + Методы

Создание и использования объектов.

Объектно-ориентированный API.

Примеры кода на JavaScript

Объектно-ориентированный подход

Класс – шаблон , на основе которого создаются объекты-экземпляры класса (с одним и тем же набором свойств и методов).

Реальный объект должен иметь конкретные значения всех полей.



*Пример описания класса и создания
объекта в PHP*

Конструкторы

Специальные методы для динамического создания в программе экземпляров класса:

- Распределяют память.
- Инициализируют атрибуты объекта.

Названия конструкторов:

- PHP: *функция `__construct()`.*
- C++, Java, C#: *совпадают с именем класса.*
- Visual Basic: *оператор New.*
- Objective-C: *оператор Init.*
- Object Pascal: *оператор Create.*

Основные принципы ООП

- **Инкапсуляция.** Пользователь, обращающийся к классу, может работать только с его интерфейсной частью и не имеет доступа к реализации функциональности класса.
- **Наследование (иерархия).** Возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка, добавляя, при необходимости, новые свойства и методы.
- **Полиморфизм.** Функции с одним и тем же именем соответствует разный программный код в зависимости от того, объект какого класса используется при вызове этой функции.

Принцип инкапсуляции

Соккрытие реализации класса и отделение его внутреннего представления от внешнего интерфейса.



Интерфейс



Реализация

Принцип инкапсуляции

Соккрытие реализации класса и отделение его внутреннего представления от внешнего интерфейса.



Интерфейс



Реализация

Принцип инкапсуляции в ООП

Доступ к данным класса возможен только посредством методов этого класса.

Свойство объекта (property) – совокупность атрибута (поля) объекта и методов его чтения/записи.



Принцип инкапсуляции в ООП

Доступ к данным класса возможен только посредством методов этого класса.

Свойство объекта (property) – совокупность атрибута (поля) объекта и методов его чтения/записи.



Профиты:

- Локализация данных, интегрирование их с подпрограммами обработки.
- Практически независимая разработка отдельных частей (объектов) программы.

Принцип инкапсуляции – реализация в ЯП

Object Pascal

Type

```
TTimePeriod = class
```

```
  function GetProperty : integer;
```

```
  procedure SetProperty(NewValue : integer);
```

```
  property Hours : integer read GetProperty write SetProperty;
```

```
End;
```

var

```
TimePeriod : TTimePeriod;
```

```
h: integer;
```

```
TimePeriod := TTimePeriod.Create;
```

```
TimePeriod.Hours := 24;
```

```
h := TimePeriod.Hours;
```


Принцип инкапсуляции - реализация в ЯП

C#

```
class TimePeriod
{
    private double seconds;
    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}
class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();
        t.Hours = 24;
        System.Console.WriteLine("Time in hours: " + t.Hours);
    }
}
```

Принцип инкапсуляции

PHP

Области видимости полей класса:

public — доступны отовсюду

private — доступны только из методов данного класса

Примеры реализации инкапсуляции в PHP

Принцип наследования/расширения

Получение по наследству атрибутов и поведений от родительских классов.

Если вас *почти* устраивает какой-то класс, вы можете создать потомка и переопределить какую-то часть его функциональности.

В С++ поддерживается **множественное наследование**, когда дочерний класс может иметь несколько родительских. Языки С#, Object Pascal, PHP этого не поддерживают.

Принцип наследования/расширения

PHP

Область видимости **protected** — поле доступно в классе, котором оно определено и во всех его дочерних классах.

Классы могут наследоваться от **абстрактного класса**, который не может иметь собственных экземпляров.

В потомке можно:

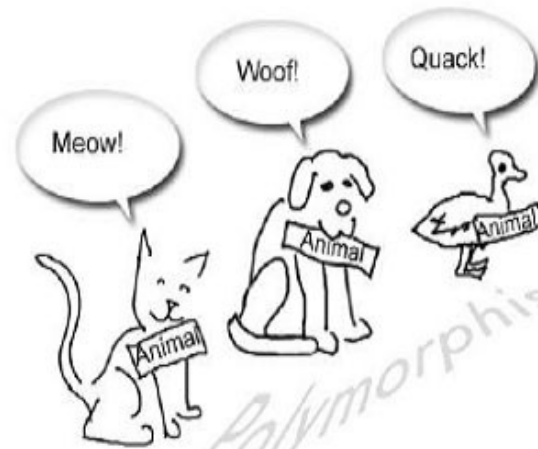
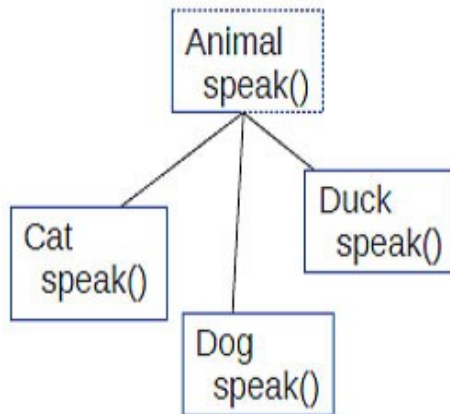
- Перекрыть метод родителя.
- Вызвать одноименный метод родителя через **parent::method()**

Пример реализации наследования на PHP

Принцип полиморфизма (позднее связывание)

Полиморфизм = "многообразие форм"

Свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.



Принцип полиморфизма (позднее связывание)

Пример реализации полиморфизма в PHP

Принцип полиморфизма (позднее связывание)

Объект подкласса (потомка) может использоваться всюду, где используется объект суперкласса (предка).

При добавлении к иерархии классов нового подкласса не нужно менять написанный код.

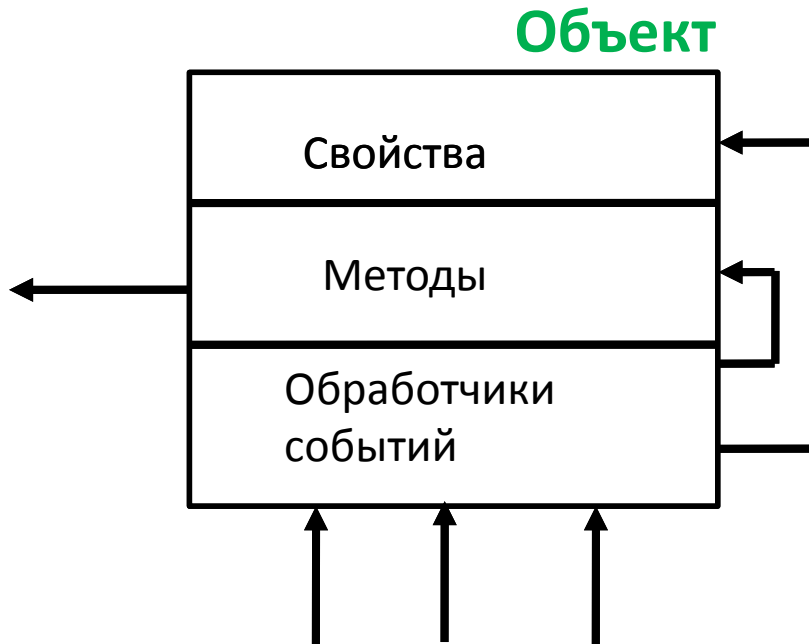
«Позднее связывание» позволяет определять версию полиморфного (виртуального) метода во время выполнения программы.

События и сообщения

Объекты могут:

- Взаимодействовать друг с другом посредством **сообщений**.
- Реагировать на определенные **события** (обрабатывать их), возникающие вследствие действий пользователя или других объектов.

Хорошо подходит для графического интерфейса (обработчики событий нажатия на кнопку и т.п.)



Тип переменной – множество отношений между данным типом и другими типами.